User Centered Design (UCD) Section

User Centered Design (UCD)

Overview

User-Centered Design (UCD) is a method of product development that involves typical users in all phases of the product development cycle. Rather than waiting until a product nears completion, user feedback is solicited continuously throughout product development. As a result, the user interface design team can make decisions based on information from real users.

UCD is Not New

Apple touted UCD in its 1989 video, *Apple World Builder*. Apple, IBM, Microsoft and many others have comprehensive web sites devoted to UCD. See the appendix for a list of helpful UCD websites.

UCD Standards

The International Standards Organization (ISO) at (http://www.iso.ch/) is developing a new standard for User-Centered Design. ISO 13407 is defined as the planning and management of a design process to facilitate the goal of making computer systems usable. It is directed mostly at project managers.

The standard elaborates on such guidance as:

- Getting a clear understanding of user and task requirements
- Employing persons from a variety of disciplines and roles in the process (e.g. user interface designers, marketing, end users, technical authors, and more)
- Practicing iterative design processes
- Evaluating designs against requirements

More recently a group of user companies launched an effort called the Common Industry Format for Usability Test Reports to create a common method for evaluating the usability of software, somewhat analogous to nutrition information and ingredients on food packages. It could save corporate software buyers millions of dollars by reducing lost productivity and/or unnecessary training from hard-to-use software.

Importance of UCD

Incorporating user-centered design principles and techniques into the process is critical to the success of the final product. Poor usability contributes to failure. A study done by the Software Engineering Institute in 1995 showed that:

- 1 in 3 software systems is cancelled
- On average, projects are 50% over schedule
- 70% are failures that don't function or aren't used

The Chaos Report

The focus of a 1995 research project (The Chaos Report) at The Standish Group identified:

1. The scope of software project failures
2. The major factors that cause software projects to fail
3. The key ingredients that can reduce project failures

The report goes on to say that US companies spend more than $250 billion each year on IT application development of approximately 175,000 projects. The average cost of a development project, broken down by size of company is:

• Large company	$2,322,000
• Medium company	$1,331,000
• Small company	$434,000

31.1% of projects will be canceled before they are completed.

52.7% of projects will cost 189% of their original estimates.

16.2% of all software projects are completed on-time and on-budget.

In large companies, only 9% of all software projects are completed on-time and on-budget.

In large companies, only 42% of the software products contain the originally-proposed features and functions.

The major factors that cause software projects to fail are also the key ingredients that can reduce project failures. The top 10 major factors, ranked by importance are:

1. User Involvement (19)
2. Management Support (16)

3. Clear Requirements (15)
4. Proper Planning (11)
5. Realistic Expectations (10)
6. Smaller Project Milestones (9)
7. Competent Staff (8)
8. Ownership (6)
9. Clear Vision and Objectives (3)
10. Hard-Working, Focused Staff (3)

The Trouble with Computers

Former director of Bell Labs, Tom Landauer, says in his book, *The Trouble with Computers,* that for every dollar spent on Information Technology, there is less than a dollar return on investment.

Software designed to make computers work well does not translate into helping people work well. If software were more intuitive and easy to use, people could spend more time doing their jobs. He predicts that if every software program were design for usability, productivity would rise by 4% to 9% annually.

The average user interface has 40 flaws. Correcting the easiest 20 yields an average improvement of 50%. The big win occurs when usability is factored in from the beginning, yielding efficiency improvements of over 700%.

Example of Time-Savings

In her book, *Cost-Justifying Usability,* Deborah Mayhew gives the following example.

Consider a simple transaction: data entry clerks filling in entries on a form. Given 20 users working 250 days a year, performing 80 transactions a day = 368,000 transactions per year. If you can reduce the time to complete the transaction by 10 seconds, you can save 1022 hours, or 25.5 person-weeks.

If you can save 1/2 a person year with improvement of one screen, improvements across the whole system will have a very dramatic effect on productivity.

Benefits of UCD

Susan Dray's article *"The Importance of Designing Usable Systems*." published in the January, 1995 edition of *interactions* listed the benefits of following a user-centered design process. They include:

- Reduced errors
- Lower support costs
- Lower initial training costs, and greatly reduced retraining

- Less productivity loss when the system is introduced, and more rapid recovery
- More focus on tasks to be done, rather than on the technology tool
- Lower turnover and better morale
- Reduced rework to meet user requirements
- High transfer of skills across applications, further reducing training needs
- Fuller utilization of system functionality
- Higher service quality
- Higher customer satisfaction
- Increased usability
- Greater user acceptance early on
- Detection of issues earlier
- Reduced documentation and support
- Increased productivity
- Reduced overall costs
- Greater sense of accomplishment for designers

UCD Precepts

The UCD is comprised of several important precepts.

- Know thy users.
- Understand the tasks the user is trying to accomplish.
- Involve the user in iterative design throughout the development process.
- Practice usability testing and usability evaluation throughout the development process.

Following these precepts doesn’t always guarantee success. Every project is different. Every failure and every success contain lessons to be learned and applied on the next project. Experience doesn’t mean that a precept can be ignored. Each project and each user is different, and every precept must be followed.

Easy is hard. If you do it right, they won’t even notice. If you do it wrong, they will all notice.